References

…

Mike Precup (mprecup@stanford.edu)
ENJMIN 2016

Reference Style

I typically write `int *x` instead of `int* x` or `int * x`

All are perfectly acceptable, but now seems like a good time to point out a C++ quirk:

```
// x, y, and z are type int

int x, y, z;

// What are they now?

int* x, y, z;
```

Reference Style

I typically write `int *x` instead of `int* x` or `int * x`

All are perfectly acceptable, but now seems like a good time to point out a C++ quirk:

```
// x, y, and z are type int

int x, y, z;

// x is int*. y and z are int.

int* x, y, z;

// You probably meant this:

int *x, *y, *z;
```

Reference Style

I typically write `int *x` instead of `int* x` or `int * x`

All are perfectly acceptable, but now seems like a good time to point out a C++ quirk:

```
// x, y, and z are type int
int x, y, z;
// x is int*. y and z are int.
int* x, y, z;
// You probably meant this:
int *x, *y, *z;
```

& and * bind to the name, not the type, so I put them next to the name

Reference Interlude

You’ve already seen functions with reference parameters:

```
void f(int &x) {
    ++x;
}

int main() {
    int x = 1;
    f(x);
    cout << x << endl; // Prints 2
}
```

Reference Interlude

You can also create references like any other variable in a function:

```
int main() {
    int x = 1;
    int& y = x;
    y = 2;
    cout << x << endl; // prints 2
}
```

Reference Interlude

Reference variables inside a function can be useful for saving a result:

```
// This function takes a long time to run
int findImportantIndex();

cout << elems[findImportantIndex()] << endl;
elems[findImportantIndex()].doThings();
elems[findImportantIndex()].add(2);
// Why is this bad?
```

Reference Interlude

We could avoid the function call by saving the computed index

```
// This function takes a long time to run
int importantIndex = findImportantIndex();

cout << elems[importantIndex] << endl;
elems[importantIndex].doThings();
elems[importantIndex].add(2);
```

Reference Interlude

- Even better, we could save the element itself
- This is faster and more concise

```
// This function takes a long time to run
Foo& important = elems[findImportantIndex()];

cout << important << endl;
important.doThings();
important.add(2);
```

Reference Interlude

Functions can also return references:

```
int global = 1;
int& getGlobal() {
    return global;
}

int main() {
    getGlobal() += 1;
    cout << global << endl; // prints 2
}
```

Reference Interlude

Functions can also return references:

```
// REALLY BAD
int& getGlobal() {
    int global = 1;
    return global;
}
```

Reference Interlude

Here’s a case where this is more useful:

```
struct Person {
    string nameVar;
    string name() { return nameVar; }
};

int main() {
    Person p = { “Joe” };
    p.name() = “Bob”; // what does this line do?
    cout << p.name() << endl;
}
```

Reference Interlude

Let’s say we add the reference:

```
struct Person {
    string nameVar;
    string& name() { return nameVar; }
};

int main() {
    Person p = { “Joe” };
    p.name() = “Bob”; // what does this line do?
    cout << p.name() << endl;
}
```

How Do References Work?

In most cases, references are based on pointers

You can think of them as pointers that are automatically deferenced

```
int x = 0;
int &y = x;
y++;

// Equivalent to
int x = 0;
int *y = &x;
(*y)++;
```